Chopper Timing

Linac LLRF

Wed, Sep 19, 2007

New hardware developed to support Linac LLRF requires that each of the five low energy Linac front ends be kept aware of the timing, especially the Chopper ON trigger times, for two or three different modes of Linac operation. The timer hardware resides in `node0602`, but nodes `0611–0615` need to know the timing. If operations modifies one of the timers, each of these front ends needs to learn of the change promptly. This note explores means of providing this general knowledge of the timer settings.

Early proposals

The simplest means of keeping track of the timer settings is for each of the five nodes to maintain an active 15 Hz request that receives the relevant settings every 15 Hz cycle, or as often as is required. (If it is ok to be one second behind, then the request might only seek 1Hz replies.) This scheme will work, but it means there is a constant up-to-75 reply transfers taking place on the network from `node0602` to the other nodes. This can work, but it makes the network activity diagnostics very busy. We may be forgiven for trying to design another method that would not require so much network communications.

Another approach involves a local application running in `node0602` that manages all of this itself. Every 15 Hz cycle, this LA would monitor its present timer settings against a list of reference values (for each of the five low energy nodes) that it maintains. If it finds a difference, it will make the appropriate adjustments for each by issuing a setting and a one-shot request (for the readback of the setting) to each node. This results in new values being passed to the timer settings that each node has for its own LLRF system. It also results in a set of replies that carry the newly-made settings back. On the next 15 Hz cycle, before `node0602` makes a determination whether another change has been made, it refreshes its reference values with each of the replied values. Normally, except for the case of an (unlikely) error in the network transaction, this will result in no change being detected, so that no more transactions ensue. A failure to get a response from a node that confirms the success of the setting, however, will result in a new attempt to deliver the latest timer setting to that node. Note that this scheme places all the burden on `node0602` to see that the other nodes are kept up-to-date on timer settings. Only one 15 Hz cycle would run in which a timer setting freshly made to `node0602` is unknown to the other nodes.

An alarm-based approach would take advantage of the multicast network delivery system that the alarm system provides. Suppose that an LA running in `node0602` to monitor changes in settings made to its timers sets a local Bit, which it clears when it sees no change. The `Alarms` task in `node0602` can then be made to deliver an alarm message about changes in this Bit. If each front end can monitor the alarm traffic, it can detect that a change was made, and when there is a change, make the necessary data requests to obtain the latest timer settings, then updating its own hardware accordingly. Note that such Bit alarm messages are not seen by Acnet.

A variation of the alarm approach would be to use not a Bit, but a Channel, in which case the timer data could be seen in the alarm message. Although such alarms would normally be seen by Acnet, we can ensure that they are not by using channels unknown to Acnet, or by some other means.

Either alarm approach requires the ability to monitor alarm message traffic. It is not immediately clear how to do this. But if it can be done, only a simple LA would have to be developed for the `node0602`, whose continued operation is fairly critical to Linac operation, since its timing hardware is used throughout the Linac.

Final approach

After due consideration for the above possible solutions, another was finally determined to be what we will install. Three timers in node0602 currently drive the chopper timing. Note that all three timers are triggered by different clock events, so only one of them fires on a given 15 Hz cycle. Assign three more timers for holding the "early warning" timers. Then OR these together in hardware, and connect the resultant signal to all five low energy Linac RF stations by cable. This means that all the timing resides in node0602, and a local application can run in that node only to ensure the appropriate alignment of the pairs of timers. Let the LA be called LLLT.

The LA is to monitor one timer, and every time it changes, presumably as a result of someone intentionally moving the trigger time, it copies the same timer delay, backed up by a constant "early warning" time, into the corresponding new timer. In this way, changing one timer causes the other to track, producing an output some constant time earlier.

With the above logic, there might be an extra cycle before the software detects that a change needs to be made. In order to avoid this, we use the following scheme. Let the channel that is changed by a user be a dummy channel; *i.e.*, its analog control field is 1200 0000. When a user changes the value of this "timer", nothing is actually written to hardware; only the setting field of the ADATA table entry is changed. Let the LA monitor this setting value. When it sees that it has changed, it acts, setting both the real hardware channel as well as the early warning channel at once. In addition, the actaul setting of the hardware channels is done at a quiet time in the Linac cycle, well after the hardware pulse has been made using the previous settings.

The local application LLLT was written to perform the above logic. Instead of watching for a change in the setting, it actually compares the reading value against the setting. Using a type 0x30 Data Access Table entry, the hardware register is read to provide the reading of not only the actual timer channel but also the dummy channel. This is why comparing the reading of the dummy channel, which is actually a copy of the hardware timer, with the setting of the dummy channel makes sense. We expect that a user will modify the dummy channel to change the chopper on time. But if a user instead changes the hardware channel, the software will notice a change (in the reading versus the setting), and it will overwrite the changed timer to what the dummy channel holds. This is thought to be desirable.

LA parameters

Param text	*Meaning*
ENABLE B	Usual enable Bit#
ANTE	Amount to set Ante timer earlier than hardware timer
USR TMR C	Dummy timer Chan#
HDW TMR C	Hardware timer Chan#
ANT TMR C	Ante (early warning) timer Chan#

Note that the ANTE parameter is a constant that must be changed via Page E. It should not be easily changed, as it must be known to the new LLRF hardware. In order to handle three different chopper ON timers, three instances of LLLT will need to be installed in node0602. The value of the ANTE parameter is in the raw units of the timer. In this case, those units are 100 ns. To make an early warning timer trigger 1 millisecond ahead of the hardware timer, use 10000, or 0x2710.

To make it easy for the users, the previous timer names will be used to target the dummy channels. New channels are created for both the hardware and the "ante" channels.

The LLLT LA also includes an internal diagnostic setting log, so we can compare a record of each timer setting that it makes in order to facilitate testing.